intel

ARTICLE REPRINT

AR-197

JANUARY, 1982

Better processor performance via global memory

JOSEPH P. ALTNETHER
Intel Corporation

ORDER NUMBER: 210378

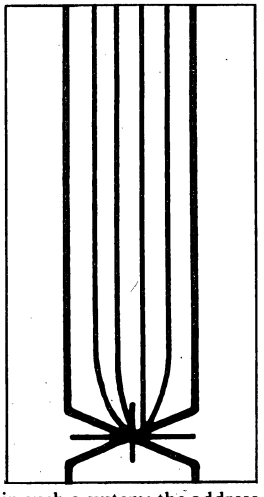
BETTER PROCESSOR PERFORMANCE VIA GLOBAL MEMORY

Wait states are eliminated by joining global and local memories through five TTL components

by Joseph P. Altnether

At least 60% of today's designs incorporate microcomputers, which have become one of the most widespread components in a variety of electronic equipment ranging from video games to navigational flight computers. Microcomputers comprise several elements. One of the more important of these is the memory. In early systems (and even in some of today's low performance microcontrollers), the memory is interfaced and accessed exactly like any other peripheral. Such an architecture is shown in Fig 1. For this type of application, data store (random access memory), control store (electrically programmable read only memory/read only memory), and input/output reside on a single bus connected directly to the central processing unit. This kind of application is usually a dedicated system performing only one function, such as control of a vending machine.

Memory consists of control store and data store. The former occupies most of the memory and contains about 16k bytes of program; the latter is small and contains less than 4k bytes. A major design goal is simplicity, which can be best achieved when the components of control store and data store are compatible. It is much simpler and certainly more efficient to use the same set of address decoders and drivers, as well as data transceivers, for both control and data store. This is achieved with common pinout and functionality between random access memory (RAM) and electrically programmable read only memory/read only memory (EPROM/ROM). Therefore, the memory should be an 8-byte wide RAM. Several disadvantages are inherent in such a system: the address space is limited; and because all elements—including the central processing unit (CPU)—reside on a common bus, the CPU, as the bus controller, suspends processing to control bus operations.



Enhancing the system

The performance of this system can be enhanced by upgrading to a microprocessor and storing a variety of programs in permanent bulk memory. In this kind of system, control store consists of a RAM containing up to 64k bytes (Fig 2). This memory is much larger because it serves a dual function: data store and control store. Programs to be executed are downloaded via a boot program residing in EPROM. The system overcomes the memory addressing space deficit of the previous system but still retains the disadvantage of having all memory

Joseph P. Altnether is technical marketing manager for memory products at Intel Corp, Aloha, OR 97005. Before that, he worked as an applications engineer at Intel and a memory system designer for nuclear/medical instruments at G. D. Searle. Mr Altnether has a BSEE from St Louis University.

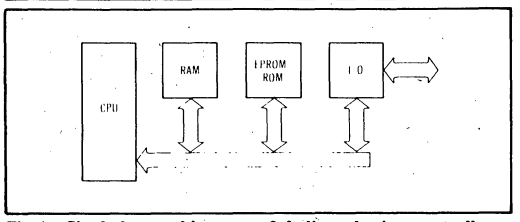


Fig 1 Single-bus architecture of dedicated microcontroller. Though inexpensive, this configuration limits available address space and requires that CPU suspend processing when controlling bus.

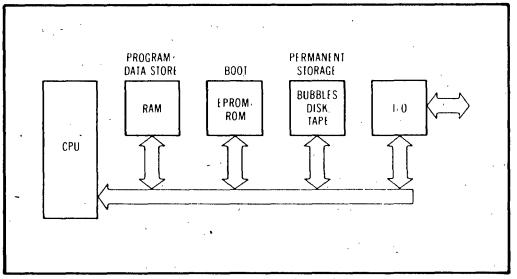


Fig 2 Improved performance results when microprocessor using RAM program storage for up to 64k bytes of data and control information is used. Disadvantages of common bus architecture are retained, however.

reside on the CPU bus. For example, throughput efficiency could be improved if it were possible to download other portions of the program into control store while executing out of control store (dual porting).

High performance in both processing power and speed is realized in distributed processing systems. In such a configuration, several processors, together with their local memories, are distributed throughout the system. These could be structured like the systems previously described; however, they have an important distinguishing element—multiple local buses with a common system or global bus. Fig 3 depicts such a system. Here, the advantages of dual porting, error checking and correction, and direct memory access all become cost effective.

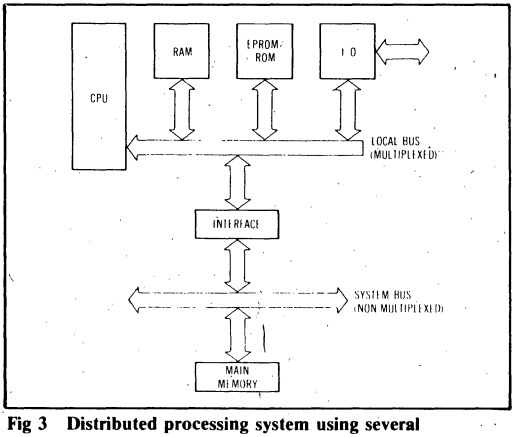


Fig 3 Distributed processing system using several processors with local memories and common (global) bus provide high performance. Each processor in system has access to large (1M-byte) global memory.

...because the global memory is so large, the RAM used must be as dense as possible to reduce the number of components.

Residing on the system bus is a global memory to which every processor has access. This memory can be very large—even greater than 1M byte. Consequently, it could be disk, tape, magnetic bubble, or RAM. If built with RAMs, the type used would be dynamic RAMs (DRAMs) for several reasons. First, because the global memory is so large, the RAM used must be as dense as possible to reduce the number of components. Lower component count reduces system cost and increases system reliability, which is inversely proportional to the number of components in the system. Second, the components should consume minimal power. Even a small amount of power per device multiplied by hundreds of devices will require a large power supply. In addition, as the power requirements increase, so do the cooling requirements, which again add to the overall system cost and operating cost.

Finally, the RAM must be low cost to be competitive and provide ample operating margins. DRAMs meet these requirements quite adequately as they provide the lowest cost per bit and also consume the lowest power per bit of RAM devices. Unfortunately, designing with DRAMs has long been considered esoteric and difficult. In fact, some designers still believe that DRAMs do not even work. The first of these beliefs was based on fact in earlier days, but the second is based on an emotional

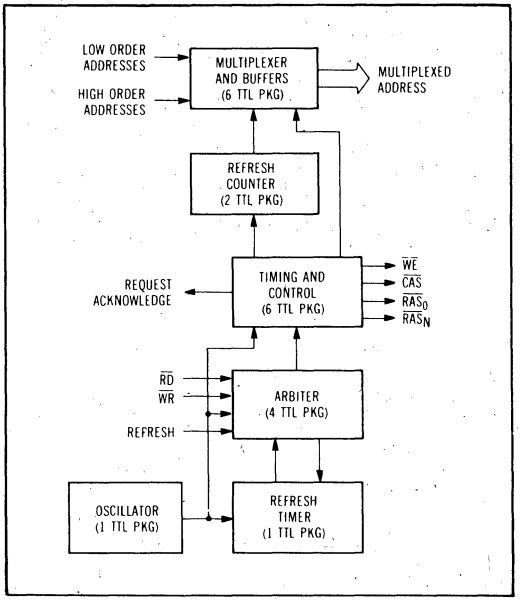


Fig 4 Typical DRAM controller. Oscillator provides timing and control logic for refresh timer.

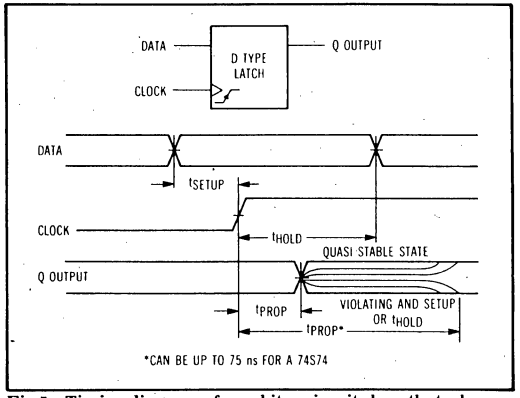


Fig 5 Timing diagrams for arbiter circuit show that when certain conditions exist, output is analog signal floating between TTL levels 1 and 0. During this 75-ns period, no decisions can be made and refresh failure occurs.

reaction to a memory that forgets unless it is periodically told to remember. DRAMs do not lose data if they are properly refreshed. This can be easily accomplished by a memory interface controller.

Designing a DRAM system

Although it is more difficult to design a DRAM system than a static RAM (SRAM) system, it is not impossible. Shown in Fig 4 is a typical DRAM controller. At the heart of the controller is an oscillator which provides timing and control logic for the refresh timer. Because DRAMs are clocked, they need signals like row address strobe (RAS), column address strobe (CAS), and write enable (WE), which come from the control logic. The refresh timer will periodically time out, typically every 15 μs, to request a refresh cycle asynchronously with respect to CPU memory requests. To decide which request (CPU or refresh) is granted first, an arbiter circuit is required. The arbiter is the most complicated controller element to design. In theory, a D type flipflop could be an arbiter (Fig 5). If refresh request is set asynchronously with respect to the system clock, a decision on the Q output can be made. If Q is true, the refresh cycle is granted; if false, the CPU is given access. Timing relationships of data and clock indicate that normal operation of the flipflop will occur if setup and hold times of data with respect to the clock are met.

If the setup or hold times are violated, however, the Q output is no longer a transistor-transistor logic (TTL) level 1 or 0. The output becomes an analog signal floating between TTL levels somewhat like a 3-state output device with the output in a high impedance state. This condition can persist for as long as 75 ns, during which it is impossible to make a decision. At the system level this appears as a refresh failure. Lastly, the controller requires multiplexers and drivers for the memory addresses. The total system is built with 20 TTL components (Fig 4).

The...DRAM controller...includes an arbiter which synchronizes the refresh and memory cycle requests to eliminate the arbitration problem....

Another consideration is design time. About four weeks are usually required for design, two weeks for worst-case analysis, six weeks for printed circuit board layout, four weeks for building and debugging, and another four weeks for redesigning to add features or correct errors. And this does not include a possible second iteration effort. In any case, the task could consume up to six man-months.

A simpler solution

Intel's dynamic RAM controller, the 8203, is contained in a single 40-pin package that incorporates the entire DRAM controller (Fig 6). It includes an arbiter which synchronizes the refresh and memory cycle requests to eliminate the arbitration problem previously described. Compatible with the 8080A, 8085A, iAPX88, and iAPX86 family of microprocessors, the device directly addresses half a megabyte of memory composed of 64k RAMs (eg, the Intel 2164). All the refresh functions are provided: timer, 8-bit address counter, and multiplexers for addresses. Because refresh is usually performed asynchronously with the CPU cycles, provision is made for performing synchronous refresh if required. At times the controller will be providing refresh when the CPU requires access. Consequently, the CPU must be placed in a WAIT mode. This is accomplished with a signal from the 8203 called SACK. In addition, the signal XACK can be used to clock data into the latches during a read cycle.

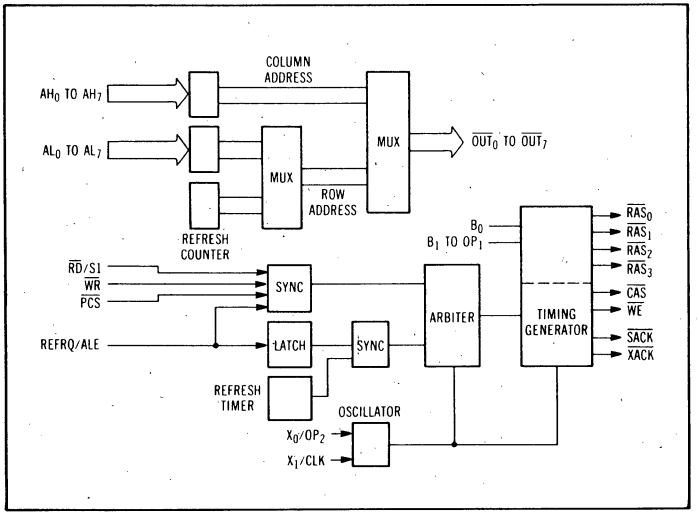


Fig 6 40-pin, 8203 DRAM controller includes arbiter that synchronizes refresh and memory cycle requests eliminating indecisive condition of Fig 5. Chip directly addresses 0.5M byte.

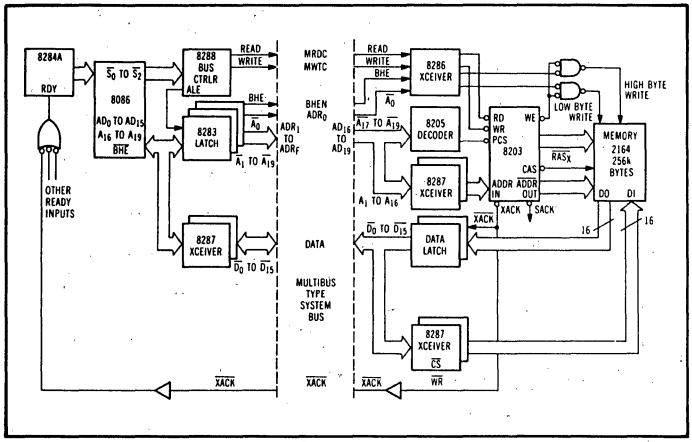


Fig 7 Typical global memory interface for microprocessor. Multiplexed address/data bus serves as local bus and demultiplexed address/data bus serves as global bus. Minimum or maximum mode operation is possible.

To illustrate the ease of interfacing global memories to the microprocessor, an iAPX86 system using an 8086 is shown. The multiplexed address/data bus is normally thought of as the local bus, and the demultiplexed address and data bus as the global bus. In much larger systems, it would be possible for the local bus to be demultiplexed immediately at the processor and for another bus that services the entire system to be the global bus. The system described works on either demultiplexed bus. The iAPX86 can be operated in either of two modes, minimum or maximum (Fig 7). In the former mode, the microprocessor generates the read and write commands directly, whereas in the latter mode, a bus controller such as the Intel 8288 is required. In this case, the iAPX86 outputs status bits S_0 to S_2 that are interpreted by the bus controller. Commands for read and write are now generated by the bus controller. Independent of the mode, the 8203 and memory interface identically to the microprocessor.

Ease of use or simplicity of design have been balanced against performance. The simple system shown (Fig 7) typically operates with one to two WAIT states required. For the minimum mode operation, the read (RD) and write (WR) commands occur too late in the memory cycle to allow the DRAM controller to generate a ready signal early enough to avoid WAIT states. Operation without WAIT states can be accomplished by transmitting advance RD or WR commands to the memory. This is a non-trivial task in the minimum mode because the iAPX86 produces the RD and WR signals in a fixed relationship after address latch enable (ALE) occurs. For maximum mode operation, the iAPX86 outputs status bit information ahead of ALE. With proper logic circuitry, these status bits can be decoded and the information used to initiate the advance RD and WR commands.

With a small amount of additional logic, it is possible to combine ease of use of the DRAM controller with high performance. As a result, the iAPX86 can operate at 5 MHz and requires no WAIT states unless the memory is

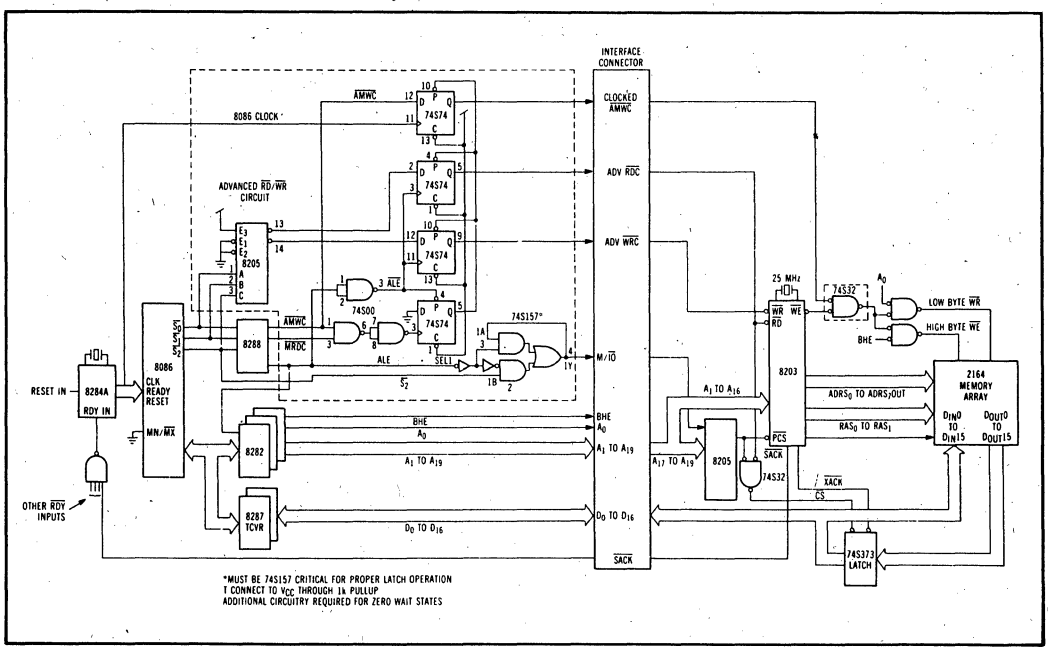


Fig 8 To achieve 5-MHz operation with no WAIT states, additional circuitry (dashed lines) must be added.

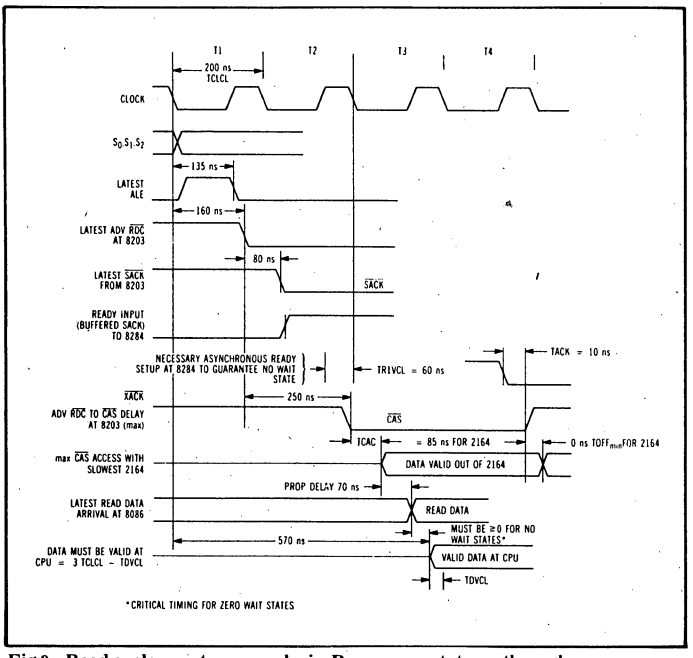


Fig 9 Read cycle worst-case analysis. Processor T states T1 through T4, are shown. For read without WAIT states, valid data must reach processor 30 ns before end of T3.

being refreshed. The circuitry added to the system is shown inside the dashed lines in Fig 8. The 8205 is a 3:8 line decoder which monitors the status lines. With the proper combination of status lines S_0, S_1, and S_2, an advanced RD command (ADV RDC) or an advanced WR command (ADV WRC) will be output on pin 13 or pin 14, respectively.

The RD or WR command, whichever is true, is latched by the corresponding 74S74 on the falling edge of ALE from the 8288 bus controller. Latch outputs at pins 5 and 9 (ADV WRC and ADV RDC) are entered into the 8203A WR and RD inputs directly. The two latches are cleared later on the trailing edge of either the memory read command (MRDC) or memory write command (MWRC) through the two 74S00 gates. System acknowledge (SACK)—used in place of (XACK) because it occurs sooner—is ANDed with protected chip select (PCS) and returned to the

Global memory can be easily built using only DRAMs and the... DRAM controller.

8284A, which provides a synchronous ready signal to the iAPX86. The S_2 status bit (memory operation) is latched by the 74S157 on the trailing edge of ALE. The 2:1 multiplexer is configured as a high speed flow-through latch by feeding the output back into the input. Propagation delay time is only 7.5 ns. The advanced memory write command (AMWC) is ANDed with WE to provide WE to the DRAMs.

Read cycle worst-case analysis (Fig 9) considers the maximum time delays. The four processor T states are indicated by T1 through T4. To read without WAIT states, valid data must reach the processor by the end of T3 minus 30 ns. The latest read data arrival at the processor does indeed fall within this time frame. The memory read cycle begins with ADV RDC (Fig 9), which is latched by the falling edge of ALE. ADV RDC reaches the 8203 at 160 ns into the cycle and begins access. Within 80 ns, SACK is valid and ANDed with PCS to be returned to the 8284A clock as READY. As a result, no WAIT states are required unless the DRAM controller is performing a refresh cycle. The system is CAS access limited, and as such the ADV RDC to CAS delay is 225 ns. The 85-ns CAS access time (t_{CAC}) must be added to this time. Finally, an additional 45-ns delay through the buffers is included for a total delay time of 510 ns. Access required is 3 T times (600 ns) minus 30 ns, or 570 ns. The system indeed requires no WAIT states for operation.

In the write cycle, the relationship between data and WE and the relationship of CAS and WE must be guaranteed. Data are written into the DRAM on the falling edge of WE. Consequently, data must be valid prior to the falling edge of WE. The skew of data from the processor and WR from the 8203 is such that it is possible for the data to be valid after the falling edge of WR. In this event, invalid data would be written into the memory as shown in Fig 10(a). In addition, DRAMs have a timing constraint, t_{CWL}, which is the overlap between CAS and WE. If CAS were early and MWTC were late, t_{CWL} would be violated as shown in Fig 10(b). Both of these requirements are satisfied by ANDing AMWC with WR.

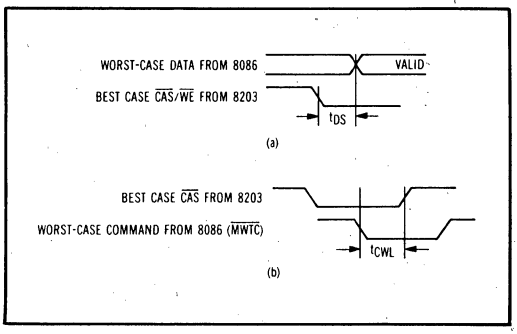


Fig 10 Required WE delay timing to memory

Fig 11 depicts the worst-case timing analysis for a write cycle, which is similar to that for the read cycle with a few exceptions. The ADV WRC is latched on the falling edge of ALE. The earliest that CAS can occur is 105 ns after ADV WRC starts the write cycle. Valid data are output from the CPU within 210 ns and reach the memory 35 ns later. By ANDing the AMWC with WR from

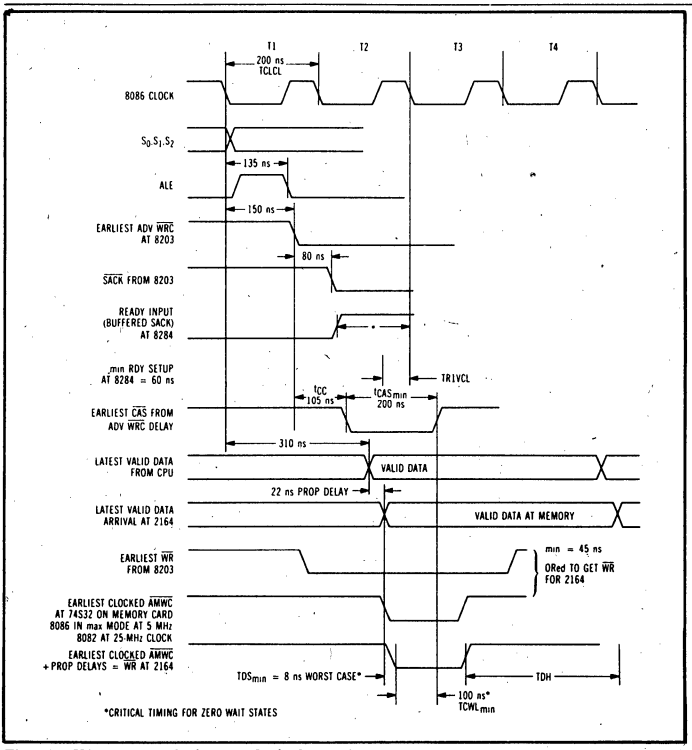


Fig 11 Worst-case timing analysis for write cycle

the 8203, WE falls a minimum of 8 ns after data are stable and valid at the memory. In addition, this ANDing guarantees a minimum t_{CWL} of 100 ns.

Overall system performance is improved by using global as well as local memories. Global memory can be easily built using only dynamic RAMs and the 8203 dynamic RAM controller. Performance, together with ease of use, is achieved by adding just five TTL components. The design of a 5-MHz system that runs without WAIT states is a good example of this approach.

Acknowledgment

The author would like to thank Bill Righter and David Chamberlin for their help in preparing this article.